SAPMA Midterm: L-System and CA Visualizations

Henry Wallace, Uthsav Chitra

March 17, 2015

1 Introduction

Our midterm project was to develop visualizations of L-systems and Cellular Automata in Python (admittedly we could have done the latter via a one-line Mathematica command, but that's no fun!). Our code is runnable from the command line, and there are multiple flags for the different options (L-system vs. CA, which L-system visualization will display, etc.). A detailed list of the flags and what they do can be found by running "python3 runner.py –help".

2 L-Systems

As a refresher, an L-system is a formal grammar. For our purposes, we can think of it as something that allows us to generalize fractals to something much easier to compute and manipulate with. There are three components to an L-system:

- the alphabet, or all possible variables appearing in any iteration;
- the axiom, or the initial starting state;
- and the production rules, which dictate how a string changes from state to state.

This definition is better served with an example: Suppose our alphabet is $\{A, B\}$, our axiom is A, and our production rules are $(A \to AB), (B \to A)$. Then, our L-system looks like:

Iteration 1: A
Iteration 2: AB
Iteration 3: ABA
Iteration 4: $ABAAB$
Iteration 5: $ABAABABA$
Iteration 6: $ABAABABAABAAB$
...

In this example, our L-system represents the famous rabbit breeding model from ecology! A is an adult rabbit, B is a child rabbit, and each successive iteration is the next generation. In this way we can take any fractal and encode it in an L-system, which allows for ease of computation. In our project, we go the other way: we take an L-system, compute the nth iteration, and draw the (a?) fractal associated with it.

We were able to plot a variety of L-systems using a very generalized function we developed. We use the turtle package (included with Python) for our plots. As you'll see, turtle can plot figures by having a cursor which creates lines as it moves around the screen. This makes it pretty simple to implement an L-system, since we can associate with each character of an L-system a specific action for the cursor to take (for instance, in our above example we could have A mean

to move forward X pixels, and B mean to turn right 90°), and then have our turtle execute the action associated with each character. Our Python script does essentially this: we make a generic function to draw a curve, given a sequence and a dictionary mapping each character to a specific action; generate the nth iteration of the L-system; and give that to our function.

Plots we will demonstrate include the Pythagoras tree, the Koch curve, the dragon curve, the Sierpinski triangle, and a fractal plant. However, our generalized function makes it easy to draw whatever fractal you want!

3 Cellular Automata

Again to refresh, a cellular automata takes an initial state on a (maybe multi-dimensional) grid, and rules for each cell depending on their current state and their neighbors. Then, it runs for many iterations, with each cell updating by the given local rule. The interaction of the cells according to this local rule can make it hard to tell what happens on a global level.

Our visualization looks at iterations of a 1-dimensional binary cellular automata, in that each state can either be "on" or "off". There are 256 rules for such a CA: each cell's state depends on its own state and its neighbors, so for a given rule, every combination of 3 cell states must be associated to a single state, yielding $2^{2^3} = 256$ possible rules.

Given a rule and an initial state, we plot what happens in each of the following iterations for n total iterations. We use matplotlib and its plotting capabilities to create these visualizations.